The Performance Evaluation Research Center (PERC)

David H Bailey

NERSC, Lawrence Berkeley National Laboratory

dhbailey@lbl.gov

PERC Overview

- An “Integrated Software Infrastructure Center” (ISIC) sponsored under DoE’s SciDAC program.
- Approximately $2.4 million per year for 4 years.
- Four DoE laboratories.
- Four universities.
- Mission:
  - Develop a science of performance, and engineer tools for performance analysis and optimization.
  - Focus on large, grand-challenge calculations, such as in SciDAC application projects.

PERC Participants

Laboratories:

- LBNL (Bailey, Strohmaier)
- LLNL (Quinlan, de Supinski, Vetter)
- ORNL (Worley, Dunigan)
- ANL (Hovland, Norris)

Universities:

- Univ of Tennessee (Dongarra)
- Univ of Illinois (Reed)
- Univ of Maryland (Hollingsworth)
- UCSD Supercomputer Center (Snavely)

Performance: Does It Matter?

Consider the economic value of improving the performance of a single high-profile, high-end scientific application code by 20%. Assume:

- $10 million computer system lease cost per year.
- $5 million per year in site costs, support staff, etc.
- 10-year lifetime of code.
- Code uses 5% of system cycles each year.

Savings: $1,500,000.

Scientific benefit probably much higher.

Feedback to Vendors

- We rely heavily on commercial vendors for high-performance computer systems.
- We are invited by vendors to provide guidance on the design of current and future systems.

BUT

- At present we can provide only vague information – little if any quantitative data or rigorous analysis.

As a result, we have little voice in future designs.

Thesis

For the foreseeable future, time to solution will be dominated by a code's ability to effectively utilize the memory hierarchy, including local and distributed memory.

Questions

- How can we best measure the memory hierarchy behavior of a particular code on a particular system?
- Can we construct accurate models of performance, based on data that is easily obtained?
- Can we accurately project the performance of a future version of a code on a future system?
- If we determine that a given code is running sub-optimally, can we facilitate the necessary changes to improve performance?

Towards a Science of Performance

- Better benchmarks.
- Performance monitoring tools.
- Performance modeling and analysis.
- Software tools to automatically or semi-automatically optimize user codes.

Benchmarks

- Indispensable for system procurements.
- Used by scientists to estimate performance of applications on present and future systems.
- Used by computer scientists to quantitatively evaluate various options in hardware, software and algorithms.

BUT in many cases,

- Problem size is too small for today's high-end systems.
- Algorithm or application is no longer meaningful.
- Doesn't precisely target the phenomenon of interest (cache behavior, communication, etc.).

Better Benchmarks

Discipline-specific benchmarks:

- Polished, concise versions of real user codes.
- Represent strategic application areas.

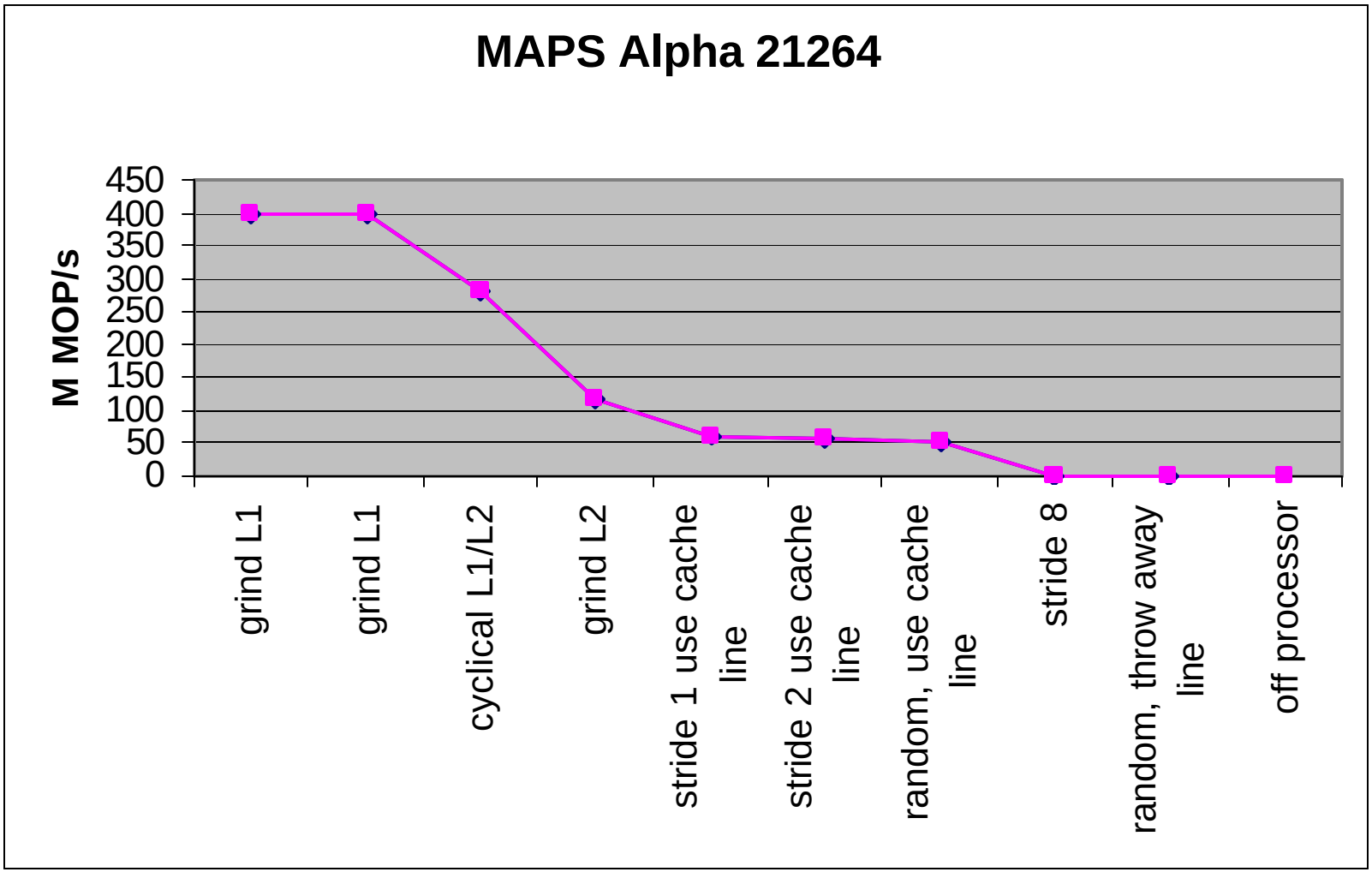
Kernel benchmarks:

- Extracted from real codes.
- Reduce complexity of analyzing full-size benchmarks.

Low-level benchmarks:

- Measure key rates of data access at various levels of memory hierarchy.
- Measure issue rates of functional units, network bandwidth and latencies, costs of TLB misses, OS context switches, I/O rates, etc.

Measured Memory Access Patterns [Alan Snaveley (SDSC)]



Performance Monitoring and Collection Tools

End-user tools:

- Integrate various analysis and measurement approaches, providing a common interface for comparing this data with benchmark and source code.

Flexible instrumentation systems:

- Capture hardware and software interactions, instruction execution frequencies, memory reference behavior, and execution overheads.

Data management infrastructure:

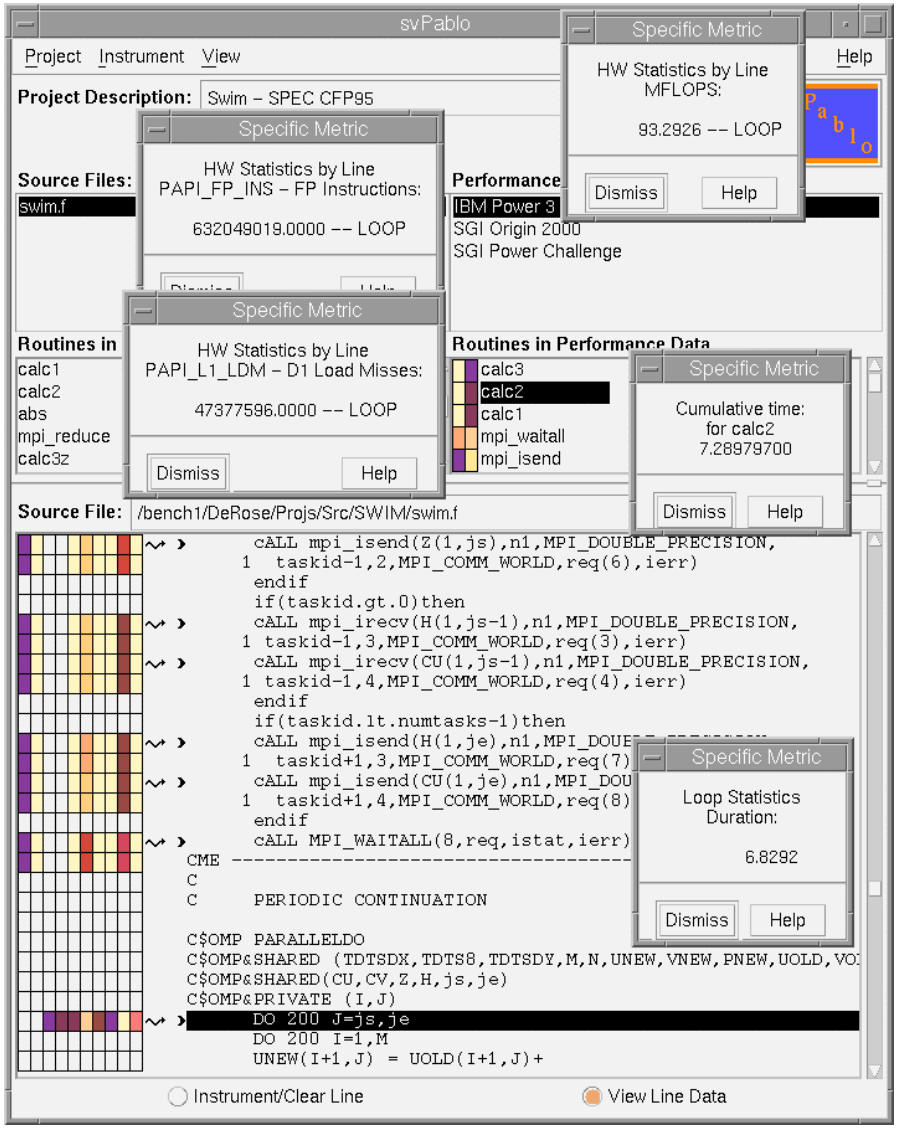
- Track performance experiments and data across time and space.

SvPablo

- A graphical environment for instrumenting application source code and browsing performance data.
- Supports performance data capture, analysis, presentation.
- C, F-77, F-90.
- Parallel and sequential systems.
- Interface to hardware performance counters, via PAPI toolkit.
- Planned enhancements:
  - Port to additional platforms: Alpha, IA-64, etc.
  - Support C++ codes.
  - Integrate with other tools.

Work is being done by Dan Reed (U Illinois) and others.

SvPablo Graphical Interface



Sigma++

- Uses runtime information to extract a detailed representation of an application's memory reference pattern.
- Post-execution tools provide insight into memory performance issues, including cache misses.
- Gathers compact address traces for regular loops.
- Planned enhancements:
  - Handle irregular references and conditional loops.
  - Statistical summary of irregular behavior.
  - Predict memory performance based on trace data.
  - Handle parallel shared memory systems.

Work is being done by Jeff Hollingsworth (U Maryland) and others.

Other Useful Performance Data Tools

- **PAPI:** A unified cross-platform tool to collect hardware performance monitor data.
- **Dyninst:** Provides a machine independent interface to permit the creation of tools and applications that use runtime code patching.
- **Repository in a box:** A toolkit to facilitate the construction and management of performance data collections.

Work is being done by Jack Dongarra (U Tennessee), Jeff Hollingsworth (U Maryland) and others.

Performance Modeling and Analysis

Holy Grail of HPC performance modeling:

- A tool and/or methodology that accurately predicts the performance of a full-scale application program, using a compact formula based on easily measurable data.

Difficulties:

- Performance function depends on many variables, including system architecture, cache design, numerical algorithms used, array sizes and others.
- Complicated interactions among variables.
- Performance function is highly non-convex – many local maxima and minima.

Analytic Phase Modeling

- Performance models based on straightforward counts of operations from source code.
- Each “phase” of the computation has its own formula.
- Advantages:
  - Reasonably accurate for some parameter ranges.
  - Quite easy to implement.
- Limitations:
  - Does not explain why rates change.
  - Does not suggest how performance can be improved.

Work is being done by Pat Worley (ORNL) and others.

Application Signatures

Characterize fundamental aspects of an application, independent of the machine where it executes.

Examples:

- Ratio of memory references to arithmetic operations.
- Memory reference patterns.
- Synchronization points.
- Instruction-level parallelism.
- Thread-level parallelism.
- Data dependencies.
- I/O characteristics.

Work is being done by Bronis De Supinski (LLNL), Jeff Vetter (LLNL), Alan Snaveley (SDSC) and others.

Machine Signatures

Characterize fundamental aspects of a machine, independent of the applications executing on it.

Examples:

- Latencies and bandwidths of memory hierarchy on local node.
- Latencies and bandwidths to remote nodes.
- Instruction issue rates.
- Cache sizes.
- TLB sizes.

Performance-predictive convolutions:

Combines application and machine signatures.

Work is being done by Alan Snaveley (SDSC) and others.

Other Modeling Techniques

- Performance algebra (Snaveley and Reed):
  - Constructs parameters for various analytic models.
- Black-box performance modeling (Strohmaier):
  - Combines generic algebraic functions with results from basic performance measurements.
- Back-fitting and statistical methods (Strohmaier, Vetter):
  - Uses sophisticated statistical methods based on empirical data.
- Performance bound modeling (Hovland).

Each of these schemes has potential advantages and disadvantages – we need to try and compare them on numerous specific problems.

Performance Optimizers

With accurate, relatively easily obtained performance information, most users can make the necessary code modifications for improved performance.

BUT:

- Project management pressures often leave little or no time for performance tuning.
- Application scientists are properly focused on their scientific discipline, not on the computer science of performance.
- Some codes are too large – automatic and/or semi-automatic tools are required.

Performance Optimizers

ROSE (Quinlan, Hovland):

- Extensible mechanism for compile-time optimization.
- Defines high-level grammars specific to user-defined abstractions.

New Harmony (Hollingsworth):

- Automatically adapts performance based on runtime observations of machine, operating environment and dataset.

ATLAS (Dongarra):

- Self-tuning linear algebra software.

AEOS (Dongarra):

- Extension of ATLAS concept to more general applications.

Performance Optimization: Other Techniques

Performance portability programming:

- Techniques to insure that code runs at near-optimal performance across a variety of modern systems.

Performance assertions:

- User-specified run-time tests that possibly change the course of the computation depending on results.
- Need a flexible, language-independent syntax.

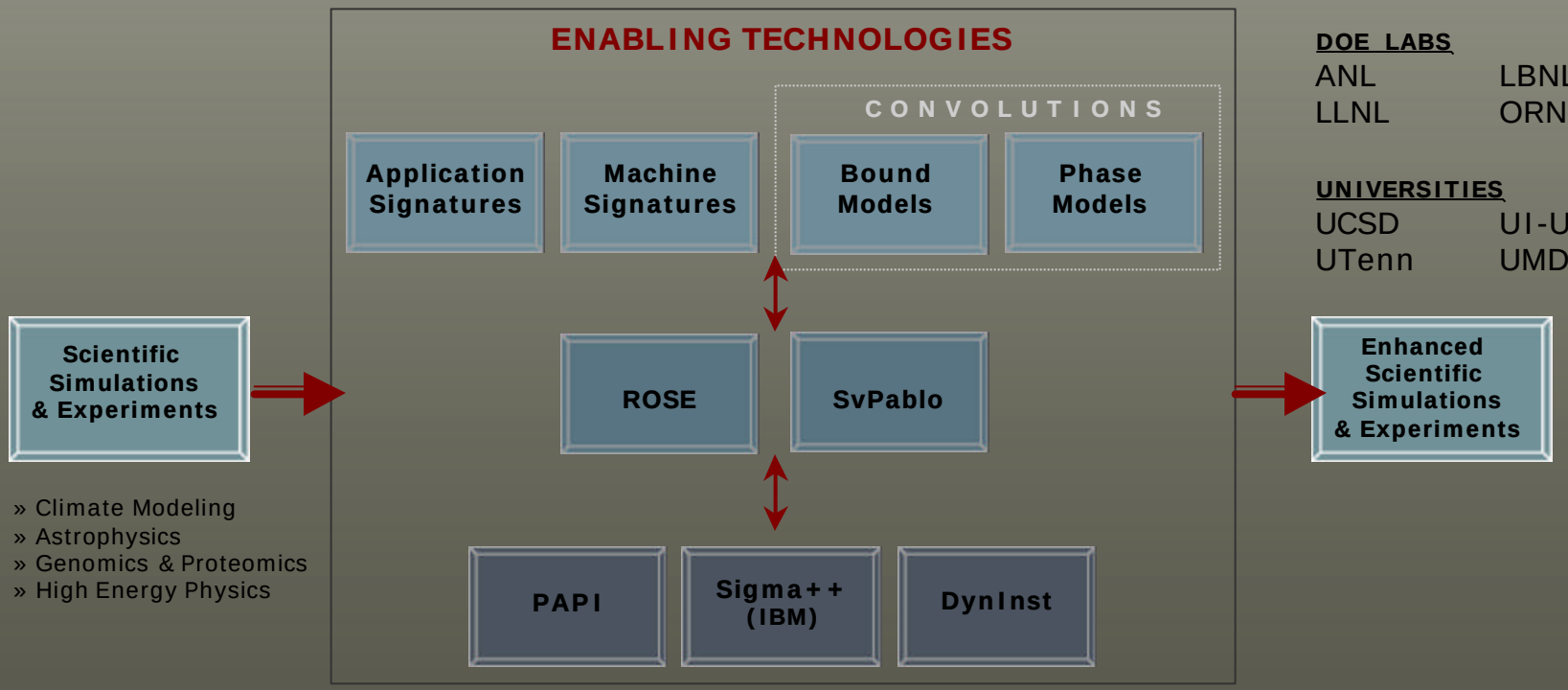
Work is being done by Pat Worley (ORNL), Jeff Vetter (LLNL) and others.

Summary

- Achieving optimal performance on HPC systems has compelling economic and scientific rationales.
- Performance is poorly understood – in depth-studies do not exist except in a handful of cases.
- PERC will pursue “performance science” and “performance engineering”, including improved benchmarks, monitoring tools, modeling techniques, and optimizers.

PERC PERFORMANCE EVALUATION RESEARCH CENTER

Developing a *science* for understanding performance of scientific applications on high-end computer systems, and *engineering* strategies for improving performance on these systems.



DOE LABS

ANL LBNL
LLNL ORNL

UNIVERSITIES

UCSD UI-UC
UTenn UMD

GOALS

Optimize and Simplify:

- Profiling of real applications
- Measurement of machine capabilities (emphasis on memory hierarchy)
- Performance prediction
- Performance monitoring
- Informed tuning

Understand the key factors in applications that affect performance.
Understand the key factors in computer systems that affect performance.
Develop models that accurately predict performance of applications on systems.
Develop an enabling infrastructure of tools for performance monitoring, modeling and optimization.
Validate these ideas and infrastructure via close collaboration with DOE SC and other application owners.
Transfer the technology to end-users.